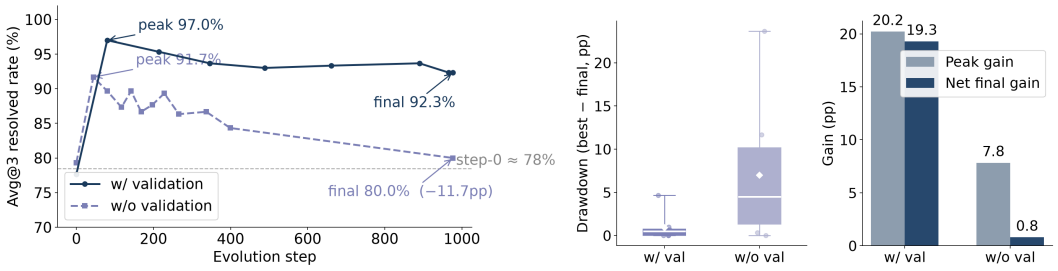
CERTIFIED LONG-HORIZON CODE AGENT EVOLUTION VIA VALIDATION-GATED SKILL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long horizon agent self-evolution without model weight updates is essential for enabling deployed agents to accumulate reusable skills and improve over time. Prior self-evolution work has focused primarily on short-horizon tasks, while repository-level software engineering remains unexplored despite being an ideal testbed for long-horizon adaptation. In this setting, agents are required to solve streams of sequential tasks, navigate complex dependencies with evolving repositories and persistently store and reuse experience. Text-based skill optimization offers an efficient, non-parametric approach for such adaptation. However, existing methods often suffer from unstable updates, performance drawdown, and agent collapse over extended deployments. In this paper, we formalize the concept of in-context self-evolution and introduce `VALVE`, a validated-gated framework for long-horizon skill optimization. We establish finite convergence, provide theoretical guarantees for future-task gain and drawdown, and derive the validation and evaluation holdout sizes required for a prescribed tolerance, with leading-order scaling $N \gtrsim s^2 L/\gamma^2$.

Empirically, our pipeline, `VALVE` achieves stable self-improvement over evolution horizon spanning more than **1,000 SWE tasks**, with average final and peak gains of 14.9 and 16.5 points across three frontier models (GPT-5.5, Claude-4.6 and MiniMax-M2.7). The validation gate reduces average drawdown by **75%** and produces an $\mathbf{11}\times$ more compact skill bank than ungated evolution. We further present extensive ablations identifying the design choices most critical to long-horizon skill evolution.



(a) Claude-4.6 evolution trajectory on `pygments`.

(b) Aggregate drawdown and retained gain.

Figure 1: **Validation stabilizes long-horizon self-evolution. (a)** Avg@3 resolved rate over approximately $1{,}000$ SWE-smith task arrivals. Both runs improve early, but the ungated run falls from a 91.7% peak to 80.0%, while the validated run peaks at 97.0% and ends at 92.3%. **(b)** Across all twelve model–repository runs, validation reduces average drawdown by 75%.

1 INTRODUCTION

Long-horizon in-context self-evolution requires an agent to continually learning throughout deployment while its model weights remain frozen. Repository-level software engineering is an ideal testbed: a code agent must process approximately sequential task arrivals (e.g., up to $1{,}000$ requests), navigate complex codebases and dependencies, and carry experience forward across diverse issues. After solving a task, it may extract reusable principles and procedures into a persistent skill bank and recall relevant items for future tasks. This form of test-time adaptation is attractive because it

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

requires neither model retraining nor changes to the solver harness. It connects prompt optimization (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024a; Yuksekgonul et al., 2024; Agrawal et al., 2026), context engineering (Zhang et al., 2026; Ye et al., 2026), skill optimization (Yang et al., 2026; Shen et al., 2026), and agent self-improvement (Zhang et al., 2025; Wang et al., 2025a), all of which adapt external text through execution feedback. However, most existing methods are evaluated in bounded optimization settings rather than over extended task streams. It remains unclear whether repeated textual updates continue to help when the same persistent memory must support hundreds of subsequent decisions (Shridhar et al., 2021; Dunn et al., 2017; Zhang et al., 2026; Ye et al., 2026).

Motivated by recent methods (Yang et al., 2026; Shen et al., 2026), we study long-horizon self-evolution through a minimal *Extract-Validate-Retrieve* formulation. Specifically, *Extract* proposes text updates based on completed task trajectories; *Validate* decides which proposals are retrained in the memory bank using held-out tasks; and *Retrieve* selects the relevant information from memory for future rollouts. This formulation exposes the central long-horizon risk. Without selective retention, individually plausible updates can accumulate, consume context and misguide future task-sovling. As shown in Figure 1b, an ungated agent may improve early yet lose much of its gain as its bank grows, sometimes finishing below its unevolved counterpart.

This formulation leads directly to a text-space optimization problem. With the LLM policy and solver harness frozen, the objective is to find a persisent memory bank that maximizes expected solve probability on the repository-level task distribution. Moreover, we further derive theoretical analyses for the corresponding optimization. First, the positive-margin gate limits the number of accepted bank changes. Under i.i.d. sampling and stated variance bounds, a disjoint evaluation set provides high-probability certificates for final population gain and peak-to-final drawdown. Because the validation set is reused and later proposals may adapt to earlier verdicts, we additionally derive a uniform population-loss bound for every accepted update. Second, we invert the same deviation bounds to determine the required holdout sizes. For a prescribed tolerance, the valdiation-panel size controls the certificated population loss of accepted updates.

We instantiate this formation in `VALVE` (**VAL**idation-gated e**V**olution for ag**E**nts) for skill optimization. On each task, the solver LLM samples a group of attempts. When their outcomes are mixed, the same LLM then contrasts successful and failed trajectories to propose a set of skills. The corresponding skill set is only accepted to the memory bank if its paired rollout on a fixed validation achieves desired improvement. An embedding-based retriever is used to independently ranks validated skills for future task solving. Thus, all tasks arrivals belong to the delopyment horizon, but only validation-approved skills are recorded in the persistent memory. To validate the effectiveness of our proposed method and theory, we carry out experiments on four SWE-smith repositories using three frontier LLMs. Compared with baselines with and without validation, `VALVE` achieves the best final gain with minimal drawdown after 1000 tasks arrivals per repository.

Our contributions are:

- **Long-horizon skill optimization for repo-level software engineering.** We introduce a pipeline that combines mixed-outcome contrastive extraction, atomic validation of joint proposals with paired executable rollouts, and selective retrieval of rich-text principles and skills. Over streams of approximately $1{,}000$ task arrivals per repository, `VALVE` achieves both best average final gain and peak gain across three frontier models and four repositories.
- **Theory for validation-gated evolution.** We establish finite convergence, provide guarantees on future-task gain and long-horizon drawdown, and derive validation and evaluation set size bounds for target performance tolerances.
- **Evidence for selective retention.** Validation reduces average drawdown from 6.1 to 1.5 points while producing an $11\times$ smaller memory bank. Component ablations and item-level analysis show how extraction, bank structure, and skill representation affect retained improvement.

2 Related Work

Context, experience, and skill optimization. A broad line of work studies how agents can improve by optimizing or accumulating external textual state while keeping model parameters fixed.

Prompt-optimization methods such as APE, ProTeGi, OPRO, EvoPrompt, TextGrad, and GEPA iteratively propose and evaluate textual updates (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024a; Guo et al., 2024; Yuksekgonul et al., 2024; Agrawal et al., 2026). Context- and experience-based methods extend this idea beyond a single prompt: ACE and MCE evolve richer context artifacts (Zhang et al., 2026; Ye et al., 2026), while Reflexion, ExpeL, AWM, ReasoningBank, and Voyager distill past trajectories into reusable feedback, strategies, workflows, or executable knowledge (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025b; Ouyang et al., 2025; Wang et al., 2023). Recent work further formalizes and constructs agent skills from trajectory evidence (Jiang et al., 2026; Li et al., 2026; Ni et al., 2026; Liu et al., 2026). Most closely related, SkillOpt treats skill text as trainable external state and accepts edits using held-out performance (Yang et al., 2026), while SkillOpt-Lite introduces an independent validation gate for skill updates (Shen et al., 2026). We study the same external-state optimization paradigm but focus on a retrieved bank of individual skills under repeated long-horizon updates, including which updates to retain and whether evolution remains stable over time.

Self-evolving code agents. A separate line of work studies self-improvement specifically in software engineering. SWE-Exp learns experience banks from repair trajectories (Chen et al., 2025), and Live-SWE-agent adapts its scaffold online (Xia et al., 2025). SICA, the Darwin Gödel Machine, the Huxley–Gödel Machine, and ARTEMIS optimize agent implementations, harnesses, or configurations through execution feedback (Robeyns et al., 2025; Zhang et al., 2025; Wang et al., 2025a; Brookes et al., 2025), while Self-play SWE-RL updates model parameters through training (Wei et al., 2026). In contrast, VALVE keeps both the LLM and solver harness fixed and isolates skill-bank evolution, allowing us to study validation, regression, and convergence over long streams of project-level software-engineering tasks.

3 VALVE: VALIDATION-GATED EVOLUTION FOR AGENTS

We first formulate long-horizon in-context self-evolution as validation-gated text-space optimization. Based on that, we then provide theoretical guarantees for performance of future tasks, including final gain and drawdown (Sec. 3.2). We further derive the holdout validation size required for a target drawdown tolerance (Sec. 3.3). Full proofs are in Appendix A.

3.1 PROBLEM SETUP AND VALVE FRAMEWORK

Problem setup. We consider an agent that sequentially encounters tasks $x_1, x_2, \ldots$ from a distribution $\mathcal{D}$ while its model weights and solver harness remain fixed. Given a task x and textual context c, the frozen policy π produces a trajectory $\tau \sim \pi(\cdot \mid x, c)$ with executable success $s(x, \tau) \in \{0, 1\}$. Its solve probability is $q(x, c) := \mathbb{E}_{\tau \sim \pi(\cdot|x,c)}[s(x, \tau)]$. The evolving state is a persistent memory bank $\mathcal{B}_t$ of textual artifacts accumulated from previous tasks. Because the full bank may consume the solver's context and even misguide its task solving, a fixed retriever R maps each task and bank into a compact context. Thus, its population performance is $F_{\mathcal{D}}(\mathcal{B}) := \mathbb{E}_{x \sim \mathcal{D}}[q(x, R(x, \mathcal{B}))]$.

A minimal framework for stable evolution. VALVE centers evolution around three operators: *Extract*, *Validate*, and *Retrieve*. Extract proposes updates from completed experience, Validate decides which updates persist, and Retrieve maps the persistent bank to a compact task-relevent solver context. For simplicity, we only consider optimization on Extract and Validate while Retrieve remains fixed. This minimal interface allows different extraction, validation, memory and retrieval mechanisms to instantiate the same framework. Figure 2 shows the specific instance for skill optimization.

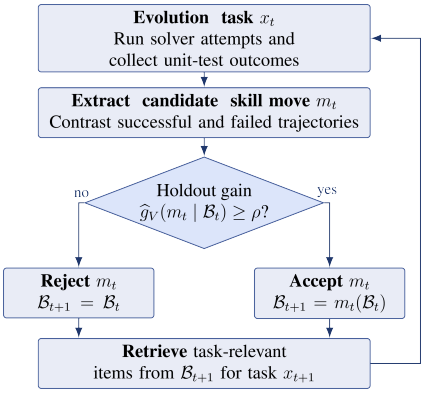


Figure 2: Overview of VALVE.

Retrieve. As Retrieve is fixed in our evolution, we first briefly discuss it. Before each task, Retrieve selects a compact subset of relevant artifacts from the current bank. We independently rank principles and skills against the issue and inject the highest-scoring rich-markdown without re-summarization (Appendix B.5).

Extract. On task x_t, the solver samples three attempts under the same task, repository and retrieved context. Extract is invoked only when executable outcomes are mixed, enabling a within-task contrast between successful and failed trajectories. A reflection call proposes at most three *principles*, which encode transferable reasoning guidance, and three *skills*, which encode narrow executable procedures. Exact and near-duplicate items are suppressed before validation (Appendices B.2–B.4).

Let m denote a proposed bank update. Its population gain is $g_{\mathcal{D}}(m \mid \mathcal{B}) = F_{\mathcal{D}}(m(\mathcal{B})) - F_{\mathcal{D}}(\mathcal{B})$.

Validate. A lesson that helps its source task may hurt others. Validate therefore compares the current and candidate banks on a held-out panel $V = \{x_1, \ldots, x_n\}$ using task-paired executable rollouts. Let F_V and g_V denote the corresponding panel quantities. A candidate is committed only when

$$\mathcal{B}_{t+1} = m_t(\mathcal{B}_t) \text{ if } \widehat{g}_V(m_t \mid \mathcal{B}_t) \geq \rho, \qquad \mathcal{B}_{t+1} = \mathcal{B}_t \text{ otherwise.} \tag{1}$$

In our implementation, each proposal jointly updates principles and skills, so the gate evaluates the complete state transition rather than individual items (Appendix B.6). For banks $\mathcal{B}, \mathcal{B}'$ and task x, we run r paired rollouts and record $\widehat{u}_{\mathcal{B}',\mathcal{B}}(x) := \frac{1}{r}\sum_{j\leq r} s(x, \tau'_j) - \frac{1}{r}\sum_{j\leq r} s(x, \tau_j) \in [-1, 1]$. Finite-rollout noise is absorbed into its variance, bounded by σ^2 on V and by $\sigma^2_{\mathcal{H}}$ on a disjoint evaluation set $\mathcal{H}$.

For an average of N i.i.d. summands with variance at most s^2 and upper deviation $Y - \mathbb{E}Y \leq M$, the one-sided Bennett radius $b_M(N, L, s^2) := \frac{s^2}{M}\,\mathrm{h}^{-1}\!\left(\frac{M^2 L}{N s^2}\right) \leq \sqrt{\frac{2s^2 L}{N}} + \frac{2ML}{3N}, \qquad \mathrm{h}(v) := (1+v)\ln(1+v) - v$ is exceeded with probability at most e^{-L}.

Assumption 1 (Sample independence)**.** For each repository, $\mathcal{D}$ denotes its task distribution. The validation panel V, held-out evaluation set $\mathcal{H}$, and adaptation stream are disjoint and i.i.d. samples from $\mathcal{D}$, and $\mathcal{H}$ is never used by extraction or validation. Candidates may depend arbitrarily on previous verdicts from V, and V may be reused throughout evolution.

3.2 Stable evolution and certified performance

Here, we show that positive-margin acceptance limits the persistent state from changing indefinitely.

Theorem 1 (Fixed-panel convergence)**.** *Suppose the panel score of the current bank is carried forward rather than re-estimated after a rejection, and let $\widehat{F}_t$ denote the panel score of $\mathcal{B}_t$ used by Equation 1. Then $\widehat{F}_t$ is non-decreasing, $\widehat{F}_T = \max_t \widehat{F}_t$, and the number K of accepted moves satisfies $K \leq \left\lfloor \frac{1-\widehat{F}_0}{\rho} \right\rfloor \leq \left\lfloor \frac{1}{\rho} \right\rfloor$. The run therefore visits at most $K+1$ distinct banks.*

The result follows directly from positive-margin ascent on a score bounded in $[0, 1]$ and holds for arbitrary history-dependent proposals and reuse of V. If improvement is localized to tasks whose baseline outcomes are unstable, with fraction h, the bound tightens to $K \leq \lfloor h/\rho \rfloor$ (Appendix A.4).

As validation convergence does not imply population improvement, we further evaluate frozen checkpoints on a disjoint set $\mathcal{H}$. Let $\mathcal{B}^{(0)}, \ldots, \mathcal{B}^{(J)}$ be the checkpoints, $\mathcal{B}^{(J)}$ the final bank, and A the number of distinct banks. Denote $\widehat{\Delta}$ for the measured final gain over $\mathcal{B}^{(0)}$, and let $\mathrm{DD} := \max_j \left[F_{\mathcal{D}}(\mathcal{B}^{(j)}) - F_{\mathcal{D}}(\mathcal{B}^{(J)})\right], \widehat{\mathrm{DD}} := \max_j \widehat{D}_j$.

Theorem 2 (Certified gain and drawdown)**.** *Under Assumption 1, with probability at least $1-\delta$,*

$$F_{\mathcal{D}}(\mathcal{B}^{(J)}) - F_{\mathcal{D}}(\mathcal{B}^{(0)}) \geq \widehat{\Delta} - b_1\big(n_{\mathcal{H}}, \ln\tfrac{1}{\delta}, \sigma^2_{\mathcal{H}}\big)\,, \tag{2}$$

$$\mathrm{DD} \leq \widehat{\mathrm{DD}} + b_1\big(n_{\mathcal{H}}, \ln\tfrac{A}{\delta}, \sigma^2_{\mathcal{H}}\big)\,, \tag{3}$$

where Equation 2 additionally requires its right-hand side to be nonnegative.

Thus a measured held-out gain certifies expected improvement on future evaluation tasks drawn from $\mathcal{D}$, while equation 3 controls peak-to-final population drawdown. Since Theorem 1 gives $A \leq K+1$, the drawdown budget depends on accepted changes rather than the total task horizon. A paired version also certifies final separation between gated and ungated runs (Theorem 4, Appendix A.6).

3.3 Validation reliability and holdout sizing

Because earlier verdicts from the same validation panel can affect later updates, we thus further bound the population loss of adaptive acceptance and then derive the required validation and evaluation set sizes.

Validation reliability. For an accepted move, define its population loss as $\ell_{\mathcal{D}}(m \mid \mathcal{B}) := [-g_{\mathcal{D}}(m \mid \mathcal{B})]_+$. Because only accepted moves change the bank, only finite states are reachable under adaptive panel reuse. Conditioning on randomness other than V, the number of bank-move pairs that may produce the k-th acceptance is at most $N_k := \sum_{t=1}^{T} \binom{t-1}{k-1}$, where T is the gate-query horizon fixed in advance.

Theorem 3 (Adaptive-validation population-loss guarantee)**.** *Under Assumption 1, let $\delta_k := \delta/(k(k+1))$ and $L_k := \ln(N_k/\delta_k)$. Then with probability at least $1-\delta$, the k-th accepted move satisfies, simultaneously for every k and every $\gamma > 0$ for which the right-hand side is at least $-\gamma$,*

$$g_{\mathcal{D}}(m \mid \mathcal{B}) \geq \widehat{g}_V(m \mid \mathcal{B}) - b_{1+\gamma}(n, L_k, \sigma^2) \geq \rho - b_{1+\gamma}(n, L_k, \sigma^2). \tag{4}$$

The guarantee is anchored at the measured validation gain and remains valid despite adaptive reuse of the same panel V.

Holdout sizing. All precedding certificates have the measured quantity where the radius decreases with the holdout size. Inverting this radius therefore gives the number of held-out tasks required for a target tolerance.

Corollary 1 (Holdout sizing)**.** *Fix $\gamma > 0$ and write $a := 2s^2L$ and $c := 2ML/3$. Then $b_M(N, L, s^2) \leq \gamma$ whenever*

$$N \geq \left(\frac{2c}{\sqrt{a + 4c\gamma} - \sqrt{a}} \right)^2 = \frac{a}{\gamma^2}\big(1 + O(\gamma)\big). \tag{5}$$

The corollary applies to the two set sizing separately. Setting $L = \ln(A/\delta)$ and $M = 1$ sizes the evaluation set for a prescribed drawdown slack γ. For the validation panel, using $L = L_K$, $M = 1 + \gamma$, and replacing γ by $\widehat{g}_V + \gamma$ ensures that every accepted update in a run with at most K accepts has population loss at most γ.

The bank count satisfies $A \leq \lfloor 1/\rho \rfloor + 1$ by Theorem 1. Baseline rollouts provide a variance proxy. If $p(x)$ is the baseline solve probability, Assumption 2 gives $\sigma^2 \leq (1 + \frac{2}{r})\mathbb{E}_x[p(x)(1-p(x))] + 2\varepsilon_{\mathrm{var}}$, whose leading term is estimated by $\mathrm{vmass} := \frac{1}{N}\sum_{i=1}^{N} \widehat{p}_i(1-\widehat{p}_i)\frac{r}{r-1}$. One can therefore estimate the variance from baseline rollouts, choose ρ, γ, and a conservative residual allowance $\varepsilon_{\mathrm{var}}$, and obtain both holdout sizes from Equation 5. Numerical examples are provided in Table 4, Figure 7, and Appendix A.9.

4 Experiments

4.1 Experimental setup

We implement `VALVE` using mini-SWE-agent (SWE-agent Team, 2026; Yang et al., 2024b), where the solver, extractor, and consumer all are based on the same backbone, using four SWE-smith repositories (Yang et al., 2025): `pygments`, `sqlfluff`, `pydicom`, and `deepdiff`. Each repository is split into disjoint partitions (Appendix B.1): a 100-task evaluation set, a fixed 50-task validation panel reused for every acceptance decision, and an evolution stream of approximately 1,000

Setting	Final resolved rate (avg@3, %)				
	`pygments`	`sqlfluff`	`deepdiff`	`pydicom`	Model Avg.
MiniMax-M2.7					
Base	52.5	66.5	56.3	60.3	58.9
SkillOpt	60.0 (+7.5)	66.0 (−0.5)	**69.0 (+12.7)**	60.7 (+0.4)	63.9 (+5.0)
VALVE w/o val	53.3 (+0.8)	66.7 (+0.2)	61.7 (+5.4)	58.7 (−1.6)	60.1 (+1.2)
VALVE	**61.0 (+8.5)**	**72.7 (+6.2)**	65.3 (+9.0)	**61.7 (+1.4)**	**65.2 (+6.3)**
GPT-5.5					
Base	59.3	68.7	64.7	38.2	57.7
SkillOpt	58.3 (−1.0)	72.0 (+3.3)	68.7 (+4.0)	56.7 (+18.5)	63.9 (+6.2)
VALVE w/o val	60.3 (+1.0)	69.7 (+1.0)	65.7 (+1.0)	68.0 (+29.8)	65.9 (+8.2)
VALVE	**62.3 (+3.0)**	**83.3 (+14.6)**	**85.0 (+20.3)**	**76.3 (+38.1)**	**76.7 (+19.0)**
Claude-4.6					
Base	78.5	79.3	71.7	71.3	75.2
SkillOpt	74.7 (−3.8)	88.7 (+9.4)	80.0 (+8.3)	**98.0 (+26.7)**	85.3 (+10.2)
VALVE w/o val	80.0 (+1.5)	62.0 (−17.3)	67.3 (−4.4)	65.3 (−6.0)	68.7 (−6.6)
VALVE	**92.3 (+13.8)**	**92.3 (+13.0)**	**97.7 (+26.0)**	96.7 (+25.4)	**94.8 (+19.6)**
Repo average					
Base	63.4	71.5	64.2	56.6	63.9
SkillOpt	64.3 (+0.9)	75.6 (+4.1)	72.6 (+8.3)	71.8 (+15.2)	71.1 (+7.1)
VALVE w/o val	64.6 (+1.1)	66.1 (−5.4)	64.9 (+0.7)	64.0 (+7.4)	64.9 (+0.9)
VALVE	**71.9 (+8.4)**	**82.8 (+11.3)**	**82.7 (+18.4)**	**78.2 (+21.6)**	**78.9 (+14.9)**

Table 1: Final performance after long-horizon SWE-smith repository-level evolution. Parentheses show gains over the corresponding base model; best results are bold.

tasks with at most 100 mixed-outcome evolution steps. All conditions use three rollouts per task, rich-markdown banks, and the acceptance threshold $\rho = 0.02$.

We compare VALVE against the recent strong skill-optimization method SkillOpt (Yang et al., 2026), VALVE without validation, and the unevolved base model. For a controlled comparison, SkillOpt and VALVE both use 50 validation tasks per repository We evaluate three frontier backone LLMs spanning different model families and capabilty levels: the open-weight MiniMax-M2.7 (MiniMax, 2026), served locally with temperature 1.0, top-p 0.95, and top-k 40, and two proprietary models, GPT-5.5 (OpenAI, 2026) and Claude-4.6-sonnet (Anthropic, 2026).

As the goal is to properly evaluate the long-horizon agent evolution, we use final-checkpoint avg@3 resolved rate on the held-out evaluation set as the primary performance metric (Table 1). To measure retention, we additionally report peak gain and peak-to-final drawdown over the evolution horizon (Table 2); complete per-repository peak results are in Appendix B.7. Applying the theorical bounds with the stated task-level variance bound $\sigma^2 = 0.025$, the 100-task evaluation set yields one-sided radii of 4.8 points for final gain and 6.5 points for drawdown at $A = 8$ (Appendix A.9). Validated evolution takes 122.1 hours per run on average, compared with 50.6 hours without the gate and 96.1 hours for SkillOpt (Appendix D).

Setting	Final gain	Peak gain	Drawdown
MiniMax-M2.7			
SkillOpt	+5.0	+7.4	**2.3**
VALVE w/o val	+1.2	+5.4	4.2
VALVE	**+6.3**	**+9.0**	2.7
GPT-5.5			
SkillOpt	+6.2	+12.8	6.6
VALVE w/o val	+8.2	+10.1	1.9
VALVE	**+19.0**	**+19.4**	**0.4**
Claude-4.6			
SkillOpt	+10.2	+12.1	2.0
VALVE w/o val	−6.5	+5.6	12.1
VALVE	**+19.6**	**+21.1**	**1.5**
Overall			
SkillOpt	+7.1	+10.8	3.6
VALVE w/o val	+0.9	+7.0	6.1
VALVE	**+14.9**	**+16.5**	**1.5**

Table 2: Gain retention over evolution. Best results are bold.

4.2 Main results

Table 1 shows that VALVE improves final performance across all three backbones. Across the twelve model-repository runs, the average resolved rate increases from 63.9% to 78.9%, a gain of 14.9 points, and every VALVE run outperforms both its unevolved base model and its counterpart without validation. The model-average gains are 6.3, 19.0, 19.6 for MiniMax-M2.7, GPT-5.5, and Claude-4.6, respectively. VALVE also outperforms SkillOpt in final model average for all three backbones, although SkillOpt remains best on MiniMax-M2.7 `deepdiff` and Claude-4.6 `pydicom`. These results are consistent with our theoretical analysis and support validaton-gated updates as a practical mechanism for reliable long-horizon agent evolution.

Validation turns transient gains into retained gains. Table 2 separates peak improvement from the gain retained at the end of evolution. Overall, SkillOpt reaches a peak gain of 10.8 points and retains 7.1, with 3.6 points of drawdown. VALVE w/o val reaches 7.0 points at peak but retains only 0.9, with 6.1 points of drawdown. VALVE instead reaches 16.5 points at peak and retains 14.9, reducing average drawdown to 1.5 points.

The contrast is strongest for Claude-4.6: without validation, the average run peaks at +5.6 points but ends at −6.5, whereas VALVE reaches +21.1 and retains +19.6. Three of the four Claude-4.6 runs without validation and the MiniMax-M2.7 `pydicom` run finish below baseline.

Figure 1 shows the same retention effect over time. On `pygments` with Claude-4.6 (Figure 1a), VALVE peaks at 97.0% and ends at 92.3% after approximately 1,000 stream tasks, whereas VALVE w/o val peaks at 91.7% and falls to 80.0%, an 11.7-point drawdown. Across all twelve runs (Figure 1b), validation reduces average drawdown from 6.1 to 1.5 points.

5 Ablation and Insights for Skill Optimization

We analyze which design choices drive skill optimization (Sec. 5.1), how validation size affects retained gains (Sec. 5.2), and how validation shapes skill-bank growth and utility (Sec. 5.3).

5.1 Component ablations

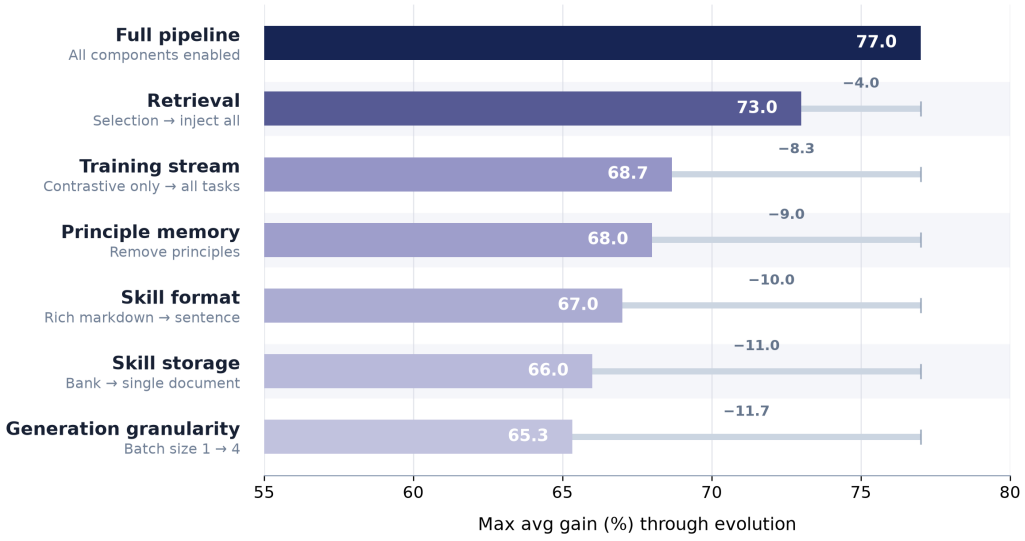


Figure 3: Ablation of component-level design choices on `pydicom` with GPT-5.5. Appendix C.1 maps prior self-evolving methods to the compared alternatives; the step-0 baseline is 38.2%.

Figure 3 compares the full pipeline with six variants, each changing one component while holding the others fixed. Both sides of every comparison correspond to design choices used in existing self-evolving methods (Table 7 in Appendix C.1 maps each choice to prior work). The full pipeline

Setting	Final bank size (skills / principles)				
	pygments	sqlfluff	deepdiff	pydicom	Total
MiniMax-M2.7					
VALVE w/o val	244/231	237/220	238/239	205/197	924/887
VALVE	30/34	38/38	14/13	34/31	116/116
GPT-5.5					
VALVE w/o val	249/194	234/160	267/210	174/127	924/691
VALVE	3/3	3/3	3/2	38/28	47/36
Claude-4.6					
VALVE w/o val	249/204	227/184	54/33	44/25	574/446
VALVE	12/14	11/10	5/4	13/12	41/40
Repo total					
VALVE w/o val	742/629	698/564	559/482	423/349	2,422/2,024
VALVE	45/51	52/51	22/19	85/71	204/192

Table 3: Final bank size after repository-level evolution. Entries report skills / principles.

outperforms every variant. Selective retrieval contributes 4.0 points, while restricting extraction to the contrastive training stream contributes 8.3 points. Removing principle memory costs 9.0 points, and replacing rich markdown with sentence-format skills costs 10.0 points. The largest drops come from replacing the skill bank with a single document (11.0 points) and increasing generation batch size from 1 to 4 (11.7 points), indicating that skill organization and extraction granularity have the largest measured effects.

5.2 Effect of validation size

Figure 4 sweeps the validation size over 0, 10, 30, and 50 tasks for Claude-4.6 on pygments. Final resolved rate rises from 80.0% without validation to 85.0%, 92.0%, and 92.3% at sizes 10, 30, and 50, with most of the improvement occurring between 10 and 30 tasks. Peak performance is not monotone: it first falls from 91.7% to 87.3%, then rises to 93.7% and 97.0%.

The final-performance trend is qualitatively consistent with Sec. 3: for fixed variance and log-budget, a larger panel shrinks the deviation radius of Theorem 3, tightening the certified bound on per-update population loss. The theorem does not imply monotone realized performance, however, because changing the panel also changes the accepted updates and the resulting evolution path. We therefore interpret this sweep as evidence that validation size affects retained gains, not as empirical verification of the bound or evidence that any particular panel size is universally sufficient.

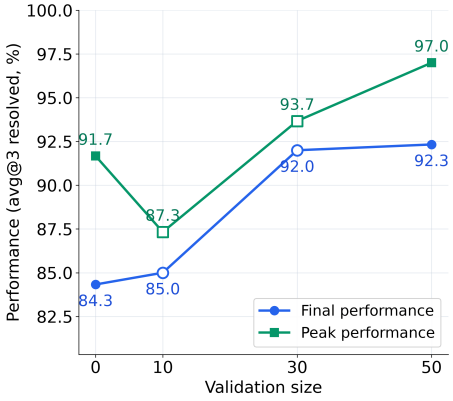


Figure 4: Final and peak avg@3 resolved rate versus validation sizes using Claude-4.6 on repository pygments.

5.3 Validated skill-bank dynamics and utility

Validation substantially changes the dynamics of bank growth. On pygments with Claude-4.6, the validated bank grows through seven accepted updates to 26 items (12 skills and 14 principles) by step 323 and then remains fixed. In contrast, VALVE w/o val accumulates 453 items by approximately step 400 yet finishes at a lower resolved rate (80.0% versus 92.3%; Figure 5). This suggests that long-horizon improvement depends not on continually accumulating more memory, but on selectively retaining useful updates.

Across all main runs, VALVE retains 204 skills and 192 principles, compared with 2,422 skills and 2,024 principles for VALVE w/o val, an 11× reduction in total bank size (Table 3).

Representative learned items

`audit_yield_tuple_index_group_consistency.md`
Code Semantics · Skill · evolved steps 323 · gain +2.00 pp.

Summarized content: Check every lexer yield tuple to ensure that its token position and value come from the same regular-expression capture group.

`enumerate_all_swapped_values_in_buggy_commit.md`
Repair Strategy · Skill · evolved steps 262 · gain +10.33 pp.

Summarized content: Inspect the complete bug-introducing commit, classify every mutation, and fix all checklist items rather than only the first visible swap.

`variable_use_before_assignment_signals_reordered_code.md`
Code Semantics · Principle · evolved steps 323 · gain +3.33 pp.

Summarized content: When a previously working function uses a variable before its later assignment, restore the intended statement order instead of adding an arbitrary default.

Figure 6: Representative learned skills and principles. Linked Appendix entries contain the full stored contents.

The retained items are also individually useful. We evaluate each of the 26 items in the validated `pygments` bank alone on the same 100-task holdout with three rollouts per task. Every item improves resolved rate over the no-memory baseline, with gains ranging from 0.33 to 10.33 points (Table 10 in Appendix E). We group the items into *Environment*, *Repair Strategy*, and *Code Semantics*, reflecting whether they concern execution, the repair process, or program behavior.

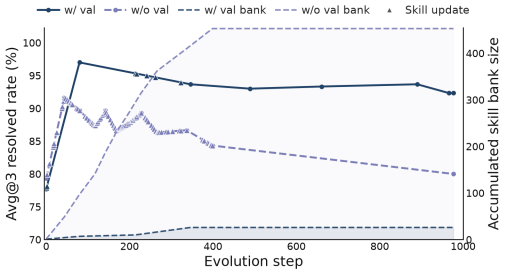


Figure 5: Accumulated bank size over evolution for Claude-4.6 on `pygments`. Solid curves show avg@3 resolved rate, dashed curves show bank size, and triangles mark bank updates.

The learned items span multiple levels of abstraction (Figure 6). Skills encode executable procedures, ranging from a Pygments-specific yield-tuple invariant to a general workflow for multi-edit regressions, while principles capture transferable diagnostic lessons, such as restoring intended statement order instead of masking the error with an arbitrary default.

6 Conclusion

Long-term self-evolution asks how an agent with a frozen LLM core can continually improve at test time, and its central obstacle is unstable updates that cause drawdown and eventual collapse over long horizons. We distilled the common design of existing in-context evolution methods into an Extract–Validate–Retrieve pipeline and formulated it as text-space optimization of a retrieved skill bank. On this formulation, positive-margin acceptance converges after finitely many bank changes, a disjoint evaluation set certifies final gain and long-horizon drawdown, and a uniform pathwise bound controls the population loss of updates chosen adaptively on a reused validation panel; inverting the same deviation radius gives sizing rules for both holdouts. Empirically, `VALVE` sustains evolution over streams of approximately 1,000 software engineering tasks per repository, achieving average final and peak gains of 14.9 and 16.5 points across three backbones and four repositories, compared with 0.9 and 7.0 points without a gate, while cutting average drawdown by 4.0× with an 11× more compact skill bank. As agents move toward lifelong deployment, this work offers practical guidance on validation gating and a step toward stable, trustworthy continual self-evolution.

Ethics Statement

All authors have read and adhere to the ICLR Code of Ethics. This work does not involve human subjects or personal data; all tasks come from the public SWE-smith benchmark, and all models are used under their respective licenses or terms of service.

Reproducibility Statement

The problem formulation, assumptions, and complete proofs are given in Section 3 and Appendix A. The `VALVE` pipeline and experimental setup are described in Section 3, and Appendix B details the configuration, seeded data partitions, prompts, acceptance rule, evaluation protocol, and ablation variants. Runtime details are in Appendix D.

AI Use Statement

We used generative AI tools to assist with part of the experiment code and to edit the paper for readability. Generative AI was not used for the theory, proofs, research ideas, experimental design, or interpretation of results. The authors have reviewed all AI-assisted content and take full responsibility for this paper.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2507.19457.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Anthropic. Claude 4.6 models. Claude Platform Documentation, 2026. URL `https://docs.anthropic.com/en/docs/about-claude/models/overview`.

Paul Brookes, Vardan Voskanyan, Rafail Giavrimis, Matthew Truscott, Mina Ilieva, Chrystalla Pavlou, Alexandru Staicu, et al. Evolving excellence: Automated optimization of LLM-based agents. *arXiv preprint arXiv:2512.09108*, 2025.

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Yong Mao, Ke Li, and Xing Sun. Training-free group relative policy optimization. *arXiv preprint arXiv:2510.08191*, 2025.

Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, and Qianxiang Wang. SWE-Exp: Experience-driven software issue resolution. *arXiv preprint arXiv:2507.23361*, 2025.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new Q&A dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.08532.

Xinmeng Hou, Peiliang Gong, Bohao Qu, Wuqi Wang, Qing Guo, and Yang Liu. Learn like humans: Use meta-cognitive reflection for efficient self-improvement. *arXiv preprint arXiv:2601.11974*, 2026.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic skills: Beyond tool use in LLM agents. *arXiv preprint arXiv:2602.20867*, 2026.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. SkillForge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026.

MiniMax. MiniMax M2.7: Early echoes of self-evolution. MiniMax, 2026. URL `https://www.minimax.io/news/minimax-m27-en`. Released March 18, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Ge Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

OpenAI. Introducing GPT-5.5. OpenAI, 2026. URL `https://openai.com/index/introducing-gpt-5-5/`. Released April 23, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. arXiv:2305.03495.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Yifei Shen, Bo Li, and Xinjie Zhang. SkillOpt-Lite: Better and faster agent self-evolution via one line of vibe. *arXiv preprint arXiv:2607.03451*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2010.03768.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

SWE-agent Team. mini-SWE-agent: The minimal software engineering agent. Software repository, 2026. URL `https://github.com/SWE-agent/mini-swe-agent`. Accessed August 3, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wenyi Wang, Piotr Piekos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley–Gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025a.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025b. arXiv:2409.07429.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward training superintelligent software agents through self-play SWE-RL. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2512.18552.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly? *arXiv preprint arXiv:2511.13646*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024a. arXiv:2309.03409.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2504.21798.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. *arXiv preprint arXiv:2601.21557*, 2026.

Shicheng Ye, Chao Yu, Kaiqiang Ke, Chengdong Xu, and Yinqi Wei. H^2R: Hierarchical hindsight reflection for multi-task LLM agents. *arXiv preprint arXiv:2509.12810*, 2025.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. arXiv:2308.10144.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2211.01910.

A Proofs and Certification Details

This appendix proves the results of Sec. 3. Appendix A.2 establishes the deviation radius and the range constants used throughout; Appendix A.3 proves convergence and Appendix A.4 the localized refinement; Appendices A.5 and A.6 prove the held-out certificates; Appendix A.7 proves the reused-panel bound; and Appendix A.8 inverts the radius.

A.1 Online dynamics of the context policy

The solve probability of a context policy μ marginalizes the context:

$$\mathbb{P}_\mu(\tau \in \mathcal{T}_x^+ \mid x) = \sum_c \underbrace{q(x,c)}_{\text{quality}} \underbrace{\mu(c \mid x, \mathcal{E})}_{\text{production probability}} = \mathbb{E}_{c\sim\mu(\cdot|x,\mathcal{E})}[q(x,c)], \tag{6}$$

where $\mathcal{T}_x^+ := \{\tau : s(x,\tau) = 1\}$. The two factors expose the central design questions behind the population objective $F_\mathcal{D}$ of Sec. 3.1: which contexts have high quality, and how should the agent produce them?

Experience is endogenous: the policy shapes the rollouts that subsequently change the policy. The agent acts causally. At step t, it observes only the strictly past experience $\mathcal{E}_{<t}$, draws $c_t \sim \mu(\cdot \mid x_t, \mathcal{E}_{<t})$, produces $\tau_t \sim \pi(\cdot \mid x_t, c_t)$, and observes $s_t := s(x_t, \tau_t)$. It then appends the complete interaction record,

$$\mathcal{E}_{<t+1} = \mathcal{E}_{<t} \parallel (x_t, c_t, \tau_t, s_t), \tag{7}$$

where $\parallel$ denotes sequence concatenation. Thus, although $F_\mathcal{D}$ (Sec. 3.1) is written as a population objective, its context policy and experience distribution are coupled through the online rollout process. Over an evolution horizon of length T, the corresponding cumulative objective is

$$\mathcal{J}_T(\mu) := \mathbb{E}\left[\sum_{t=1}^T s_t\right] = \mathbb{E}\left[\sum_{t=1}^T q(x_t, c_t)\right], \tag{8}$$

where the expectation includes task sampling, context selection, and trajectory generation. A successful evolution rule must therefore improve future contexts without using information from future interactions.

A.2 The deviation radius

Lemma 1 (One-sided Bennett radius)**.** *Let $Y_1, \ldots, Y_N$ be i.i.d. with mean ν, variance at most s^2, and $Y_i - \nu \le M$ almost surely. Then*

$$\mathbb{P}\left(\frac{1}{N}\sum_{i=1}^N Y_i - \nu > b_M(N, L, s^2)\right) \le e^{-L}, \tag{9}$$

where b_M is the one-sided Bennett radius of Sec. 3.1,

$$b_M(N, L, s^2) := \frac{s^2}{M}\,\mathrm{h}^{-1}\!\Big(\frac{M^2 L}{N s^2}\Big) \le \sqrt{\frac{2s^2 L}{N}} + \frac{2ML}{3N}, \qquad \mathrm{h}(v) := (1+v)\ln(1+v) - v. \tag{10}$$

Proof. Bennett's inequality for independent summands with the stated variance and one-sided range gives, for every $b > 0$,

$$\mathbb{P}\left(\frac{1}{N}\sum_{i=1}^N Y_i - \nu > b\right) \le \exp\left(-\frac{Ns^2}{M^2}\,\mathrm{h}\Big(\frac{Mb}{s^2}\Big)\right), \qquad \mathrm{h}(v) = (1+v)\ln(1+v) - v. \tag{11}$$

The function h is strictly increasing on $[0,\infty)$ with $\mathrm{h}(0) = 0$, so it is invertible there. Setting the exponent in equation 11 equal to $-L$ and solving,

$$\frac{Ns^2}{M^2}\,\mathrm{h}\Big(\frac{Mb}{s^2}\Big) = L \iff \frac{Mb}{s^2} = \mathrm{h}^{-1}\Big(\frac{M^2 L}{Ns^2}\Big) \iff b = \frac{s^2}{M}\,\mathrm{h}^{-1}\Big(\frac{M^2 L}{Ns^2}\Big), \tag{12}$$

which is the first expression in equation 10.

For the displayed relaxation, use $\mathrm{h}(v) \geq v^2/(2+\frac{2}{3}v)$, valid for $v \geq 0$. Substituting $v = Mb/s^2$ turns equation 11 into

$$\exp\left(-\frac{Ns^2}{M^2}\cdot\frac{M^2b^2/s^4}{2+\frac{2Mb}{3s^2}}\right) = \exp\left(-\frac{Nb^2}{2s^2+\frac{2M}{3}b}\right). \tag{13}$$

Setting this to e^{-L} gives $Nb^2 = L\big(2s^2+\frac{2M}{3}b\big)$, that is $b^2 \leq a' + c'b$ with $a' = 2s^2L/N$ and $c' = 2ML/(3N)$. Since $b^2 \leq a'+c'b$ implies $b \leq \sqrt{a'}+c'$, we obtain $b \leq \sqrt{2s^2L/N}+2ML/(3N)$. □

Every summand below lies in $[-1,1]$, for which $M = 2$ is always admissible. The following remark identifies the smaller admissible range in each of our statements.

Remark 1 (Effective range)**.** *Each certificate has the form: establish $\nu \geq c$ for a population contrast ν, by showing that the empirical mean does not exceed ν by more than b, with c equal to the empirical value minus b. On the failure event $\nu < c$, the deviation required is*

$$\frac{1}{N}\sum_i Y_i - \nu \;\geq\; b + (c-\nu), \tag{14}$$

which grows as ν decreases, while the admissible range $M = 1-\nu$ also grows. Substituting both into the exponent of equation 11 gives

$$\frac{Ns^2}{(1-\nu)^2}\,\mathrm{h}\!\left(\frac{(1-\nu)(b+c-\nu)}{s^2}\right). \tag{15}$$

For completeness, this exponent is nondecreasing in $-\nu$ whenever, for every $\nu \in [-1,c]$,

$$\frac{(1-\nu)(b+c+1-2\nu)}{s^2}\ln(1+v_\nu) \;\geq\; 2\,\mathrm{h}(v_\nu), \qquad v_\nu := \frac{(1-\nu)(b+c-\nu)}{s^2}. \tag{16}$$

This follows by differentiating the displayed exponent with respect to $1-\nu$ and is directly checkable for any reported (N, s^2, b, c). We verified equation 16 over the full interval for the numerical configurations used in this paper. Under this condition the binding case is $\nu \to c$ and the effective range is $M = 1-c$. Consequently: for the drawdown equation 3, only checkpoints with a positive drop can attain the maximum, so $c = 0$ and $M = 1$; for the gain equation 2 and the separation equation 32, $c \geq 0$ whenever the certified floor is nonnegative, so $M = 1$; and for the reused panel equation 4, $c = -\gamma$ and $M = 1+\gamma$. If equation 16 is not verified, the same statements remain valid with the conservative range $M = 2$, because every summand lies in $[-1,1]$.

A.3 Proof of fixed-panel convergence

Proof of Theorem 1. A rejected move leaves both the bank and its recorded panel score unchanged, so $\widehat{F}_{t+1} = \widehat{F}_t$. An accepted move at step t replaces the current bank by $m_t(\mathcal{B}_t)$ and, because the accepted candidate's score is carried forward, replaces $\widehat{F}_t$ by the candidate's score. Hence

$$\widehat{F}_{t+1} - \widehat{F}_t = \widehat{g}_V(m_t \mid \mathcal{B}_t) \;\geq\; \rho, \tag{17}$$

the inequality being the acceptance rule equation 1. In both cases $\widehat{F}_{t+1} \geq \widehat{F}_t$, so the sequence is non-decreasing and $\widehat{F}_T = \max_t \widehat{F}_t$.

Let $t_1 < \cdots < t_K$ be the accepted rounds. Summing equation 17 over them and telescoping,

$$\widehat{F}_{t_K+1} - \widehat{F}_0 = \sum_{j=1}^{K}\big(\widehat{F}_{t_j+1} - \widehat{F}_{t_j}\big) = \sum_{j=1}^{K}\widehat{g}_V(m_{t_j} \mid \mathcal{B}_{t_j}) \;\geq\; K\rho. \tag{18}$$

Since $\widehat{F}_t$ is an average of quantities in $[0,1]$ it satisfies $\widehat{F}_t \in [0,1]$, whence

$$K\rho \;\leq\; \widehat{F}_{t_K+1} - \widehat{F}_0 \;\leq\; 1 - \widehat{F}_0 \;\leq\; 1, \tag{19}$$

and dividing by ρ and taking integer parts gives equation 1. The bank changes only at accepted rounds, so the number of distinct banks visited is at most $K+1$.

The argument is deterministic. It uses no property of V beyond F_V being bounded, and in particular permits arbitrary reuse of V and history-dependent proposals. □

Remark 2. *If the current bank were re-scored with fresh rollouts at every comparison instead of carried forward, equation 17 would hold only up to the re-estimation drift. On the event that every such drift is bounded by $\eta < \rho/2$, the same telescoping gives $K \leq \lfloor 1/(\rho-2\eta)\rfloor$.*

A.4 BASELINE-CONCENTRATED HEADROOM

Let $p(x)$ denote the baseline solve probability, let $U := \{x : p(x) \in (0,1)\}$ be the baseline-unstable set, and let $h := \mathbb{P}_{x\sim\mathcal{D}}(x \in U)$. With only r baseline rollouts, define the observed unstable panel fraction $\widehat{h} := n^{-1}\sum_{i=1}^{n} \mathbf{1}\{0 < \widehat{p}_i < 1\}$. This is generally downward biased for h: tasks with small but nonzero $p(x)$ can produce only failures in all r rollouts.

Assumption 2 (Baseline-concentrated gain)**.** There are residual allowances $\varepsilon_{\mathrm{loc}}, \varepsilon_{\mathrm{var}} \geq 0$ such that every reachable bank satisfies

$$F_V(\mathcal{B}) - F_V(\mathcal{B}_0) \leq \widehat{h} + \varepsilon_{\mathrm{loc}}, \tag{20}$$

and every reachable move satisfies

$$\mathbb{E}_x[u_{m,\mathcal{B}}(x)^2] \leq \mathbb{E}_x[p(x)(1-p(x))] + \varepsilon_{\mathrm{var}}, \tag{21}$$

$$\mathbb{E}_x[\mathrm{Var}(\widehat{u} \mid x)] \leq \frac{2}{r}\mathbb{E}_x[p(x)(1-p(x))] + \varepsilon_{\mathrm{var}}. \tag{22}$$

Under Assumption 2, the number of accepted moves satisfies the localized bound

$$K\rho \;\leq\; \widehat{h} + \varepsilon_{\mathrm{loc}}, \qquad \text{i.e.,} \qquad K \;\leq\; \left\lfloor \frac{\widehat{h} + \varepsilon_{\mathrm{loc}}}{\rho} \right\rfloor. \tag{23}$$

Proof of equation 23. Assumption 2 directly bounds the realizable panel headroom. Using it in the telescoping step of Appendix A.3 gives $K\rho \leq \widehat{h} + \varepsilon_{\mathrm{loc}}$, which is equation 23. The idealized zero-residual result is recovered at $\varepsilon_{\mathrm{loc}} = 0$, but we do not impose that value on the experiments. □

The variance part of the assumption and the law of total variance give

$$\mathrm{Var}\left[\widehat{u}\right] = \mathrm{Var}_x\left[u\right] + \mathbb{E}_x\big[\,\mathrm{Var}(\widehat{u} \mid x)\big] \;\leq\; \Big(1 + \frac{2}{r}\Big)\mathbb{E}_x\big[p(x)(1-p(x))\big] + 2\varepsilon_{\mathrm{var}}. \tag{24}$$

For r independent Bernoulli baseline rollouts on task i, the plug-in estimator satisfies

$$\mathbb{E}\big[\widehat{p}_i(1-\widehat{p}_i)\big] = \frac{r-1}{r}\,p_i(1-p_i), \tag{25}$$

so multiplying by $r/(r-1)$ removes the finite-r downward bias, and averaging over tasks gives equation 3.3 as an unbiased estimate of $\mathbb{E}_x[p(1-p)]$. When $r = 3$ each empirically unstable task has $\widehat{p} \in \{1/3, 2/3\}$ and contributes $1/3$, so empirically $\mathrm{vmass} = \widehat{h}/3$. This identity does not make $\widehat{h}$ an unbiased estimate of the population mass h.

A.5 PROOF OF THE CERTIFIED GAIN AND DRAWDOWN

Proof of Theorem 2. We prove the two statements separately.

Final gain. The reporting rule fixes the pair $(\mathcal{B}^{(0)}, \mathcal{B}^{(J)})$ before $\mathcal{H}$ is drawn: both banks are measurable functions of the adaptation stream, the validation panel and the remaining randomness of the run, none of which involve $\mathcal{H}$ under Assumption 1. Conditioning on all of that randomness, the per-task summands

$$Y_i := \widehat{u}_{\mathcal{B}^{(J)},\mathcal{B}^{(0)}}(x_i'), \qquad x_i' \in \mathcal{H}, \tag{26}$$

are i.i.d. in $[-1, 1]$ with

$$\mathbb{E}[Y_i] = F_{\mathcal{D}}(\mathcal{B}^{(J)}) - F_{\mathcal{D}}(\mathcal{B}^{(0)}) =: \nu, \qquad \mathrm{Var}[Y_i] \leq \sigma_{\mathcal{H}}^2, \tag{27}$$

and their average is $\widehat{\Delta}$. Applying Lemma 1 with $L = \ln(1/\delta)$ and, by Remark 1, $M = 1$ whenever the certified floor is nonnegative,

$$\mathbb{P}\Big(\widehat{\Delta} - \nu > b_1\big(n_{\mathcal{H}}, \ln\tfrac{1}{\delta}, \sigma_{\mathcal{H}}^2\big)\Big) \leq \delta, \tag{28}$$

and on the complementary event $\nu \geq \widehat{\Delta} - b_1$, which is equation 2. No union bound arises because the comparison is fixed before $\mathcal{H}$ is seen.

Drawdown. Let $D_j := F_{\mathcal{D}}(\mathcal{B}^{(j)}) - F_{\mathcal{D}}(\mathcal{B}^{(J)})$ with paired estimate $\widehat{D}_j$ on $\mathcal{H}$. Checkpoints that hold the same bank give identical D_j and identical $\widehat{D}_j$, so it suffices to index the A distinct banks. Each pair $(\mathcal{B}^{(j)}, \mathcal{B}^{(J)})$ is again fixed before $\mathcal{H}$ is drawn, because the checkpoint schedule is chosen in advance and the banks are functions of the stream and the panel. Applying Lemma 1 to each with $L = \ln(A/\delta)$ and $M = 1$ by Remark 1, and taking a union bound over the A indices,

$$\mathbb{P}\Big(\exists j:\ D_j > \widehat{D}_j + b_1\big(n_{\mathcal{H}}, \ln\tfrac{A}{\delta}, \sigma^2_{\mathcal{H}}\big)\Big)\ \le\ A \cdot \frac{\delta}{A} = \delta. \tag{29}$$

On the complementary event, taking the maximum over j on both sides,

$$\mathrm{DD} = \max_j D_j\ \le\ \max_j \widehat{D}_j + b_1 = \widehat{\mathrm{DD}} + b_1, \tag{30}$$

which is equation 3. That $M = 1$ is admissible follows from Remark 1 with $c = 0$: since $D_J = 0$, both $\mathrm{DD} \ge 0$ and $\widehat{\mathrm{DD}} \ge 0$, so indices with $D_j \le 0$ satisfy the conclusion trivially and only $D_j > 0$ needs covering. Finally, $A \le K + 1$ by Theorem 1. □

A.6 Certified separation between gated and ungated evolution

Write $\mathcal{B}^{\mathrm{val}}$ and $\mathcal{B}^{\mathrm{noval}}$ for the final banks produced on a given repository with and without the gate.

Theorem 4 (Certified separation of retained performance)**.** *Fix a repository with task distribution $\mathcal{D}$, evaluate both runs on the same held-out set $\mathcal{H}$, and let σ^2_W bound the variance of the per-task cross-condition contrast*

$$W(x) := q\big(x, R(x, \mathcal{B}^{\mathrm{val}})\big) - q\big(x, R(x, \mathcal{B}^{\mathrm{noval}})\big). \tag{31}$$

Then with probability at least $1 - \delta$,

$$F_{\mathcal{D}}(\mathcal{B}^{\mathrm{val}}) - F_{\mathcal{D}}(\mathcal{B}^{\mathrm{noval}})\ \ge\ \widehat{W} - b_1\big(n_{\mathcal{H}}, \ln\tfrac{1}{\delta}, \sigma^2_W\big). \tag{32}$$

Corollary 2 (Repository average)**.** *If the statement is made for G repositories with disjoint evaluation sets of $n_{\mathcal{H}}$ tasks each, then with probability at least $1 - \delta$ the average of the G repository-level differences exceeds the average of the $\widehat{W}_r$ by at most $b_1\big(G\, n_{\mathcal{H}}, \ln\frac{1}{\delta}, \sigma^2_W\big)$.*

The contrast is paired: both runs are scored on the same tasks, so task difficulty cancels within each task and one radius bounds the difference.

Proof of Theorem 4. Both final banks are measurable functions of their own adaptation streams, the validation panel of the gated run, and the remaining randomness, none of which involve $\mathcal{H}$ under Assumption 1. Conditioning on that randomness, the per-task values

$$W(x'_i) = q\big(x'_i, R(x'_i, \mathcal{B}^{\mathrm{val}})\big) - q\big(x'_i, R(x'_i, \mathcal{B}^{\mathrm{noval}})\big), \qquad x'_i \in \mathcal{H}, \tag{33}$$

are i.i.d. in $[-1, 1]$ with mean $F_{\mathcal{D}}(\mathcal{B}^{\mathrm{val}}) - F_{\mathcal{D}}(\mathcal{B}^{\mathrm{noval}})$ and variance at most σ^2_W, and their empirical average is $\widehat{W}$. Lemma 1 with $L = \ln(1/\delta)$ and $M = 1$ gives equation 32.

For the corollary, index repositories by $r = 1, \dots, G$. Their evaluation sets are disjoint, so the per-task values are independent across repositories as well as within them, and the grand average

$$\frac{1}{G}\sum_{r=1}^{G} \widehat{W}_r = \frac{1}{G n_{\mathcal{H}}} \sum_{r=1}^{G}\sum_{i=1}^{n_{\mathcal{H}}} W_r(x'_{r,i}) \tag{34}$$

is an average of $G n_{\mathcal{H}}$ i.i.d. bounded summands whose mean is the average of the G repository-level differences. Applying Lemma 1 with $N = G n_{\mathcal{H}}$ gives the claim. □

A.7 Proof of the reused-panel population-loss guarantee

Condition on the randomness ξ of the run other than the validation panel: the adaptation task stream, solver sampling, extraction sampling and retrieval tie-breaking. For an accept pattern $S \subseteq [T]$ define recursively

$$\mathcal{B}(S, 1) = \mathcal{B}_0, \qquad m(S, t) = \mathrm{Prop}\big(\xi, \mathcal{B}(S, t), t\big), \qquad \mathcal{B}(S, t+1) = \begin{cases} m(S,t)\big(\mathcal{B}(S,t)\big), & t \in S, \\ \mathcal{B}(S,t), & t \notin S. \end{cases} \tag{35}$$

Lemma 2 (Path compression)**.** *For every S and t: (i) $\mathcal{B}(S,t)$ depends on S only through $S \cap [t-1]$; (ii) the pair $(\mathcal{B}(S,t), m(S,t))$ is ξ-measurable; (iii) the realized run is the path of $S^* = \{\text{accepted steps}\}$; and (iv) for each $k \geq 1$ the family*

$$\mathcal{F}_k(\xi) := \Big\{ \big(\mathcal{B}(S,t), m(S,t)\big) : t \in [T_{\text{gate}}],\ S \subseteq [T_{\text{gate}}],\ t \in S,\ |S \cap [t-1]| = k-1 \Big\} \tag{36}$$

of pairs that can occur at an accepted step preceded by exactly $k-1$ accepts satisfies $|\mathcal{F}_k(\xi)| \leq N_k$ with N_k as in equation 3.3.

Proof. (i) The recursion equation 35 consults S only at indices strictly below t, so $\mathcal{B}(S,t)$ is unchanged if S is modified at indices $\geq t$. (ii) By induction on t: $\mathcal{B}(S,1) = \mathcal{B}_0$ is fixed, and if $\mathcal{B}(S,t)$ is ξ-measurable then so is $m(S,t) = \text{Prop}(\xi, \mathcal{B}(S,t), t)$, the proposal mechanism being a deterministic function of ξ, the current bank and the step index; hence $\mathcal{B}(S,t+1)$ is ξ-measurable. (iii) Immediate from the definition of the gate equation 1, which accepts exactly at the steps in S^*. (iv) A pair in $\mathcal{F}_k$ is determined by the step t at which it occurs together with the prefix $S \cap [t-1]$, by (i). The constraint $t \in S$ and $|S \cap [t-1]| = k-1$ means the prefix is a subset of $[t-1]$ of size exactly $k-1$, of which there are $\binom{t-1}{k-1}$. Summing over $t \in [T_{\text{gate}}]$ gives

$$|\mathcal{F}_k(\xi)| \ \leq\ \sum_{t=1}^{T_{\text{gate}}} \binom{t-1}{k-1} = N_k. \tag{37}$$

□

The family $\mathcal{F}_k(\xi)$ is defined without reference to the panel: it enumerates what the optimizer could have produced under every possible sequence of verdicts. Each of its elements is therefore a fixed bank-and-move pair against which the panel is an honest i.i.d. sample.

Proof of Theorem 3. Fix $k \geq 1$ and work conditionally on ξ. Let $(\mathcal{B}, m) \in \mathcal{F}_k(\xi)$. By Lemma 2(ii) it is ξ-measurable and hence, by Assumption 1, independent of the panel. The panel values

$$Y_i := \widehat{u}_{m(\mathcal{B}),\mathcal{B}}(x_i), \qquad x_i \in V, \tag{38}$$

are therefore i.i.d. in $[-1,1]$ with mean $g_{\mathcal{D}}(m \mid \mathcal{B})$ and variance at most σ^2, and their average is $\widehat{g}_V(m \mid \mathcal{B})$. Lemma 1 with $L = L_k$ and, by Remark 1, $M = 1+\gamma$, gives

$$\mathbb{P}\Big(\widehat{g}_V(m \mid \mathcal{B}) - g_{\mathcal{D}}(m \mid \mathcal{B}) > b_{1+\gamma}(n, L_k, \sigma^2)\Big) \ \leq\ e^{-L_k} = \frac{\delta_k}{N_k}. \tag{39}$$

A union bound over the at most N_k elements of $\mathcal{F}_k(\xi)$ gives

$$\mathbb{P}\Big(\exists (\mathcal{B}, m) \in \mathcal{F}_k(\xi) :\ \widehat{g}_V - g_{\mathcal{D}} > b_{1+\gamma}(n, L_k, \sigma^2)\Big) \ \leq\ \delta_k. \tag{40}$$

By Theorem 1 the run accepts at most $\lfloor 1/\rho \rfloor$ moves surely, so only finitely many strata are nonempty, and summing equation 40 over $k \geq 1$,

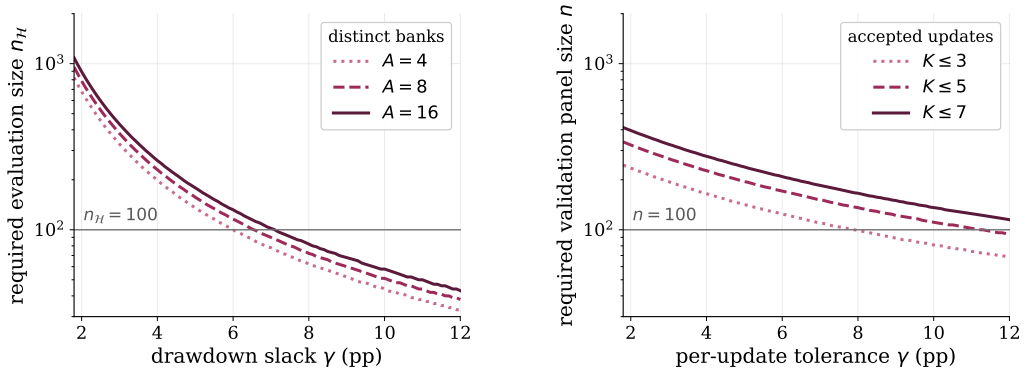
$$\sum_{k \geq 1} \delta_k = \delta \sum_{k \geq 1} \frac{1}{k(k+1)} = \delta \sum_{k \geq 1} \left(\frac{1}{k} - \frac{1}{k+1} \right) = \delta. \tag{41}$$

Hence with probability at least $1-\delta$ the event in equation 40 fails for every k simultaneously. On that event, the k-th accepted move of the realized run lies in $\mathcal{F}_k(\xi)$ by Lemma 2(iii)–(iv), so

$$g_{\mathcal{D}}(m \mid \mathcal{B}) \ \geq\ \widehat{g}_V(m \mid \mathcal{B}) - b_{1+\gamma}(n, L_k, \sigma^2) \ \geq\ \rho - b_{1+\gamma}(n, L_k, \sigma^2), \tag{42}$$

the last step by the acceptance rule equation 1. The bound holds conditionally on ξ with probability at least $1-\delta$ for every ξ, and therefore unconditionally. □

The proof uses no independence between candidates, no freshness of the panel and no restriction on how proposals react to past verdicts; reusing the rollouts of the unchanged current bank induces dependence across the elements of $\mathcal{F}_k(\xi)$, which a union bound tolerates. It does require T_{gate} to be fixed in advance, since it determines N_k.



(a) Sizing the evaluation set.

(b) Sizing the validation panel.

Figure 7: Holdout size required for a prescribed tolerance, from Corollary 1. The vertical axes are logarithmic; horizontal lines mark the 100-task evaluation set and the 50-task validation panel used in our runs.

	Evaluation set $n_{\mathcal{H}}$			Validation panel n		
Tolerance γ	$A = 4$	$A = 8$	$A = 16$	$K \le 3$	$K \le 5$	$K \le 7$
2pp	680	787	895	235	324	396
3pp	328	380	432	195	268	327
5pp	136	157	179	142	195	239
8pp	62	72	82	99	136	166
10pp	44	51	58	81	111	136

Table 4: Holdout size required for a prescribed tolerance, at $\delta = 0.05$ and $\sigma^2 = \sigma_{\mathcal{H}}^2 = 0.025$. Left: tasks needed so that the certified population drawdown exceeds the measured one by at most γ, for A distinct banks among the checkpoints. Right: tasks needed so that every accepted update of a run making at most K of them costs no more than γ, at a gate-query horizon $T = 60$ and anchored at the observed accepted panel gain of 5.8 points.

A.8 Proof of the sizing rule

Proof of Corollary 1. By Lemma 1 it suffices to make the displayed relaxation of b_M at most γ, that is

$$\sqrt{\frac{a}{N}} + \frac{c}{N} \;\le\; \gamma, \qquad a := 2s^2 L, \quad c := \frac{2ML}{3}. \tag{43}$$

Substituting $u := N^{-1/2} > 0$ turns this into the quadratic condition

$$c\,u^2 + \sqrt{a}\,u - \gamma \;\le\; 0, \tag{44}$$

whose left-hand side is increasing in u on $u > 0$ and vanishes at

$$u^\star = \frac{-\sqrt{a} + \sqrt{a + 4c\gamma}}{2c}. \tag{45}$$

The condition is therefore $u \le u^\star$, i.e. $N \ge (u^\star)^{-2}$, which is equation 5. Expanding $\sqrt{a + 4c\gamma} = \sqrt{a}\,(1 + 2c\gamma/a + O(\gamma^2))$ gives $u^\star = \gamma/\sqrt{a} + O(\gamma^2)$ and hence $N = (2s^2 L/\gamma^2)(1 + O(\gamma))$, the stated leading order. All reported sizes invert the exact radius b_M rather than its relaxation, and are therefore no larger. □

A.9 Numerical instantiation

Parameters: $\rho = 0.02$, $\delta = 0.05$, $r = 3$, $n = 50$, $n_{\mathcal{H}} = 100$, and $G = 4$ repositories with disjoint evaluation sets. The protocol caps gate queries at one per triggering task with at most 100 such

tasks; across the main runs the gate was queried 492 times in total, an average of 61.5 per run with a maximum of 85, and a horizon of $T = 60$ is used for prospective sizing. The realized number of accepted updates averages 4.75 with a maximum of $K = 7$, so the localized bound $\lfloor h/\rho \rfloor = 10$ of equation 23 is not binding. Accepted updates clear the threshold with a run-level minimum averaging 5.8 points and a run-level median of 7.1 points.

Theorem 1 also gives the sharper run-specific cap $\lfloor (1 - \widehat{F}_0)/\rho \rfloor$. The formal cap uses the initial validation-panel score of each run. For scale, substituting the held-out step-0 rates in Table 1 gives caps of $20, 15, 17, 30$ for the four GPT-5.5 runs and $10, 10, 14, 14$ for the four Claude-4.6 runs, respectively, rather than the distribution-free cap $\lfloor 1/\rho \rfloor = 50$. These values are illustrative because the held-out rates are not the panel scores used by the gate; the exact certificate uses the recorded $\widehat{F}_0$ for each run.

Pooling $24{,}600$ task-level candidate-minus-current differences gives mean -0.006184 and variance 0.024954, rounded to $\sigma^2 = 0.025$; the baseline unstable fraction averages $h = 0.2075$, giving $\mathrm{vmass} = 0.069$ from equation 3.3. Two further variance bounds enter the certificates: $\sigma^2_{\mathcal{H}}$, taken equal to σ^2, and σ^2_W, taken as 0.05; the certificates are stated as holding under these bounds.

At these values the radii are 4.8 points for the final gain, 6.5 points for the drawdown at $A = 8$, and 3.0 points for the four-repository separation, so the measured 0.92, 17.2 and 8.0 points certify $\mathrm{DD} \le 7.5$, a gain $\ge +12.4$ and a separation $\ge +5.0$ points respectively. Inverting the radius, a 10-point drawdown tolerance requires 51 evaluation tasks at $A = 8$, and the same tolerance on every accepted update of a run making at most seven of them requires 136 panel tasks at $T = 60$.

B Implementation details

B.1 Data construction and reference configuration

We use one common Extract–Retrieve–Validate protocol across repository-level experiments; only the repository, backbone, or explicitly ablated component changes. Table 5 summarizes this reference protocol rather than a configuration specialized to any single repository.

Component	Reference setting
Data	One repository-level task stream from the SWE-smith training split
Evaluation set	100 tasks sampled without replacement (seed 42)
Validation panel	50 tasks sampled from the remaining stream (seed 123); the same fixed panel is used for every acceptance decision
Agent	Frozen code-agent backbone (MiniMax-M2.7, GPT-5.5, or Claude-4.6-sonnet, according to the run) with an interactive shell environment
Adaptation	3 independently sampled rollouts per task
Evolution trigger	Mixed-outcome tasks only; at most 100 mixed tasks
Persistent state	Separate rich-text principle and skill banks
Retrieval	Task-conditioned top-10 principles and top-10 skills
Validation	Paired candidate/current comparison on 50 tasks, 3 rollouts per condition and task, acceptance threshold $\rho = 0.02$
Evaluation	100 held-out tasks at step 0 and every 10 mixed tasks, with 3 rollouts per task

Table 5: Reference configuration shared by the main repository-level runs.

Repository selection. We study DeepDiff, Pygments, pydicom, and SQLFluff, selected from the high-volume SWE-smith repositories using two practical criteria. First, each provides more than $1{,}000$ nonempty tasks to construct substantial adaptation, validation, and evaluation partitions, and yields reliable patch-producing agent runs. Second, they exercise markedly different forms of repository knowledge: recursive object comparison, syntax highlighting, medical-image processing, and SQL parsing and linting. This combination lets us test whether persistent skills help across distinct APIs, test structures, and debugging patterns rather than within variants of one application domain.

Within each repository, the partition is constructed before adaptation. We first sample and remove the evaluation set, then reserve 50 validation tasks, and use the remainder as the adaptation stream.

Thus no exact instance appears in more than one partition. The 50 reserved tasks form one fixed validation panel reused at every comparison rather than a fresh sample at each update. The prepared order of the adaptation stream is preserved, and independent fixed seeds make the partitions reproducible.

B.2 Rollout collection and evolution trigger

For each adaptation task x_t, the agent receives the issue description and an interactive repository environment in which it can inspect files, edit source code, and execute tests. An attempt terminates when the agent submits a patch or reaches its interaction budget. The submitted patch is evaluated by the executable task-specific test harness; this outcome, rather than the agent's self-assessment, supplies the binary label.

Before generating attempts, Retrieve selects one memory context M_t from the current bank. Three rollouts are then sampled independently under the same issue and frozen context. Let $R = 3$ and let $z_{t,r} \in \{0, 1\}$ indicate whether attempt r passes the executable tests. We call task t *mixed* when

$$0 < \sum_{r=1}^{R} z_{t,r} < R. \tag{46}$$

Mixed tasks provide a within-task contrast: the issue, repository state, and retrieved context are held fixed, but the sampled trajectories contain both successes and failures. The full pipeline applies Extract only to this subset, avoiding lessons inferred solely from uniformly successful or uniformly failed experience. All tasks are nevertheless solved and scored. The evolution index counts mixed tasks, so the 100-step budget can require observing approximately $1{,}000$ raw tasks.

B.3 Extract: contrastive reflection

Extract maps the labeled rollout set $\tau_t = \{(h_{t,r}, p_{t,r}, z_{t,r})\}_{r=1}^{R}$ to a candidate update $\Delta_t = (\Delta_t^P, \Delta_t^S)$, where h is the interaction history and p the submitted patch. The reflection context contains the issue, the three labeled summaries, their final results, and the names of existing memories. The latter discourages paraphrased duplicates. Extract returns at most three principles and three skills.

We distinguish the two memory types semantically. A *principle* captures a domain-agnostic reasoning strategy—how to reason under uncertainty. A *skill* captures a narrow, executable software-engineering procedure, search strategy, testing pattern, or recurring mistake to avoid. The central system instruction used for same-task contrastive extraction is shown below; formatting examples are omitted for space.

> **System prompt: contrastive Extract**
>
> *You are a learning agent extracting transferable lessons from multiple attempts at the same software engineering task. Return strict JSON with exactly three keys: issue_summary, principles, skills.*
>
> *Focus on: what reasoning distinguished successful attempts from failed ones; mistakes repeated across failed attempts; concrete techniques that worked.*
>
> **Principles:** *meta-cognitive, domain-agnostic reasoning strategies. Principles describe HOW to think and reason, not what to do procedurally. They must be useful beyond any single language, framework, or task type.*
>
> **Skills:** *specific, actionable software-engineering techniques. They must be narrow enough to apply immediately and grounded in what went wrong or what worked in the trajectories.*
>
> *Output only valid JSON. Return at most three new principles and three new skills; return empty lists if there are no new insights worth adding.*

The accompanying user message instantiates the following template:

User prompt template

Task: [issue description]
This task was attempted R times: [number] succeeded and [number] failed.

Attempt 1 [SUCCESS/FAILED]: [trajectory summary; final result]
...
Attempt R [SUCCESS/FAILED]: [trajectory summary; final result]

Already known principles/skills (do not repeat or closely paraphrase): [item names]. Extract at most three new principles and three new skills.

Each principle contains a short search sentence and a structured body with **Statement** and **When to apply**. Each skill contains a search sentence, a scope hint, and a structured body with **Scope**, **Trigger**, and a numbered **Procedure**. Optional anti-patterns and code examples make the learned text directly usable by a future solver. The logical output schema is

```
{
  "principles": [{"name": "...", "text": "...",
                  "markdown": "..."}],
  "skills": [{"name": "...", "text": "...",
              "scope_hint": "...", "markdown": "..."}]
}
```

Malformed outputs produce an empty proposal rather than an unconstrained repair attempt. This conservative choice prevents malformed reflections from entering the persistent state.

B.4 Persistent skill state

In the reference configuration, the state $M = (P, S)$ consists of separate append-only banks of principles P and skills S. An item stores its name, search sentence, rich body, source tasks, support count, creation step, usage count, and source-outcome statistics; skills additionally retain a scope hint. These fields support both retrieval and later inspection of provenance.

Before validation, exact normalized duplicates are merged with their existing items. A candidate is also suppressed when its token-set Jaccard similarity to an existing item is at least 0.45:

$$J(a, b) = \frac{|T_a \cap T_b|}{|T_a \cup T_b|} \geq 0.45. \tag{47}$$

Here T includes the search sentence and rich body, as well as the scope hint for skills. The reference configuration performs no periodic summarization or LLM-based consolidation, ensuring that accepted updates remain individually traceable.

B.5 Retrieve: task-conditioned skill composition

Retrieve independently ranks principles and skills for the current issue. The query combines the task identifier and issue description. For item i, let Q denote query tokens, C_i the item's search-text tokens, and M_i its metadata tokens (the scope hint for a skill and the empty set for a principle). The score is

$$\begin{aligned} s(i, q) = &\underbrace{\frac{|Q \cap (C_i \cup M_i)|}{\min(|Q|, |C_i \cup M_i|)}}_{\text{overlap coefficient}} + \underbrace{\sum_{w \in Q \cap C_i} \log_2 \max\left(\frac{N}{df(w)}, 1\right)}_{\text{text IDF}} \\ &+ 0.45 \underbrace{\sum_{w \in Q \cap M_i} \log_2 \max\left(\frac{N}{df(w)}, 1\right)}_{\text{scope IDF}} + 0.5 I_i + \cos(e_q, e_i), \end{aligned} \tag{48}$$

where N is the size of the corresponding bank. The lexical terms reward direct issue and scope matches, while cosine similarity captures semantic relatedness between the issue and rich item body.

I_i is an importance prior derived from source-task outcomes, with support count as a fallback. Semantic vectors use the Qwen3-Embedding-0.6B model when available, with a deterministic hashing representation as a fallback. The ten highest-scoring principles and ten highest-scoring skills are selected independently.

The selected rich bodies are composed into the solver context rather than re-summarized. This preserves procedures, triggers, and anti-patterns exactly as validated. The injected system-prompt block has the following form:

System-prompt augmentation produced by Retrieve

Learned principles:
— [principle name] —
[Statement; when to apply; optional anti-pattern]
...

Learned skills:
— [skill name] —
[Scope; trigger; numbered procedure; optional anti-pattern]
...

Use the above when relevant, but do not follow them blindly if repository evidence contradicts them.

The final instruction treats retrieved memory as defeasible guidance rather than an authority that can override repository evidence.

B.6 VALIDATE: PAIRED ACCEPTANCE ON EXECUTABLE OUTCOMES

Extract proposes a *joint* bank move containing all new principles and skills from the current mixed task. Validate compares two conditions on the same fixed 50-task validation set:

- **current**: task-conditioned retrieval from the current bank M;
- **candidate**: the same retrieved context augmented by all items in Δ_t.

Each condition receives three independently sampled and executably graded rollouts per task. For $c \in \{\mathrm{cur}, \mathrm{cand}\}$, the empirical pass rate is

$$\widehat{p}_c = \frac{1}{50} \sum_{j=1}^{50} \frac{1}{3} \sum_{r=1}^{3} z_{j,r}^{(c)}. \tag{49}$$

All reported experiments use the acceptance threshold $\rho = 0.02$. The proposed move is committed exactly when

$$\widehat{p}_{\mathrm{cand}} \geq \widehat{p}_{\mathrm{cur}} + 0.02. \tag{50}$$

Thus each condition aggregates 150 binary outcomes, and the threshold requires at least three net additional successful rollouts on the $1/150$ score grid. The comparison validates the joint state transition rather than attributing utility to individual items. Results for the unchanged current state are reused until an accepted update changes the bank, reducing repeated evaluation without changing the acceptance rule.

The same 50 tasks are reused for every candidate. They are 50 distinct task draws with three within-task replications, not 150 independent population-task samples. Accordingly, $\rho = 0.02$ is an empirical acceptance margin rather than a confidence level. A comparison uses up to $2 \times 50 \times 3 = 300$ executable rollouts before reuse of the unchanged current result. Across $24{,}600$ task-level paired samples from the main runs, the candidate-minus-current difference has mean -0.006184 and variance 0.024954, rounded to $\sigma^2 = 0.025$. Theorem 1 characterizes fixed-panel convergence with score carry-forward, while Theorem 3 gives the corresponding uniform population-loss guarantee over the paths induced by panel reuse. Table 4 and Figure 7 report the resulting sizing rules; the numerical instantiation is detailed in Appendix A.9.

B.7 Evaluation protocol and reported statistic

Before adaptation and every ten mixed tasks, we freeze the current bank and evaluate it on a disjoint 100-task panel. Each task receives three independent rollouts. For panel size $H = 100$, avg@3 is

$$\text{avg@3} = 100 \times \frac{\sum_{j=1}^{H} \sum_{r=1}^{3} z_{j,r}}{3H}. \tag{51}$$

The denominator includes all $3H$ attempts, including empty or unsuccessful submissions. The main table reports the final-checkpoint avg@3 first and the highest observed avg@3 over the trajectory in parentheses. The latter uses this panel for checkpoint selection and therefore is not an untouched final-test estimate. When gains are discussed, final and peak gains are computed relative to the shared step-0 evaluation. We retain the bank state, extracted proposals, rollout histories, executable verdicts, validation comparisons, and periodic evaluation results needed to reconstruct each accepted transition.

B.8 Ablation implementations and prompts

Table 6 describes each intervention at the method level. Because some interventions jointly alter representation and control flow, we state these couplings explicitly rather than treating every command-line change as a perfectly isolated causal operator.

Ablation	Method-level intervention
Sentence skills	Replace structured rich bodies with one-sentence principles and scoped skill sentences throughout extraction, storage, and injection.
No principles	Constrain Extract to return an empty principle set; retrieve and inject skills only.
Single document	Replace itemized banks with one principle document and one skill document, updated through section-level add/replace/delete operations and injected in full. This also removes item-level retrieval.
Inject all	Remove task-conditioned ranking and inject every stored item in reverse creation order.
Batch of four	Buffer four completed tasks and perform one cross-task reflection; use one validation rollout per validation task.
All tasks	Apply Extract and Validate after all-pass and all-fail tasks as well as mixed tasks; the 100-step horizon consequently counts raw tasks.

Table 6: Method-level definitions of the ablations.

Batch extraction. For the batch-size ablation, Extract receives four task blocks, each containing the issue, outcome label, trajectory summary, and final result. Principles must recur across at least two tasks or be sufficiently universal to justify retention from one instance; skills may remain task-specific when reusable. The core system instruction is:

> **System prompt: batched Extract**
>
> *You are a learning agent that extracts transferable lessons by comparing multiple completed software engineering tasks.*
>
> *Principles must appear as a recurring pattern across at least two tasks in this batch, OR be so universally applicable that a single instance justifies inclusion. Skills can be task-specific as long as they are genuinely reusable.*

In mixed-only mode, a batch is reflected upon when it contains at least one mixed task, but neighboring all-pass or all-fail examples remain visible to the cross-task reflection. Since this condition also uses one validation rollout per task, it changes both extraction granularity and validation noise.

Single-document extraction. The single-document condition represents each memory type as sections under named headings. Reflection proposes bounded section edits of the form

Single-document update schema

operation: add, replace, or delete
target: existing section name
name: proposed section name
text: one-sentence summary
markdown: complete replacement body
scope hint: where or when a skill applies

A replacement removes the target section and appends the proposed version; a missing replacement target becomes an addition, while an addition that reuses an existing name is ignored. Because the resulting documents are always injected in full, this condition jointly changes extraction, storage, and retrieval rather than merely changing the on-disk format.

Interpretive scope. The remaining couplings are similarly important. All-task evolution changes both the extraction trigger and the number of raw tasks observed before stopping; removing validation also removes validation-derived reflection feedback; and high reasoning changes reflection as well as solving. Ablation differences should therefore be interpreted as effects of these operational method variants, not as estimates of perfectly isolated primitives.

C Additional ablation and skill-bank details

C.1 Prior work behind each ablated component

Table 7 maps both sides of every component ablation in Figure 3 to existing self-evolving methods.

Component	Prior work with our choice	Prior work with the ablated choice
Retrieval Selection → inject all	Voyager (Wang et al., 2023), ReasoningBank (Ouyang et al., 2025), Memp (Fang et al., 2025)	AWM (Wang et al., 2025b), ExpeL (Zhao et al., 2024), SkillOpt (Yang et al., 2026)
Training stream Contrastive only → all tasks	ExpeL (Zhao et al., 2024), H^2R (Ye et al., 2025), Training-Free GRPO (Cai et al., 2025)	ReasoningBank (Ouyang et al., 2025), ACE (Zhang et al., 2026), SkillOpt (Yang et al., 2026)
Principle memory Remove principles	MARS (Hou et al., 2026), SkillRL (Xia et al., 2026), H^2R (Ye et al., 2025)	Voyager (Wang et al., 2023), AWM (Wang et al., 2025b)
Skill format Rich markdown → sentence	Trace2Skill (Ni et al., 2026), EvoSkill (Alzubi et al., 2026), SkillOpt (Yang et al., 2026)	ExpeL (Zhao et al., 2024), ACE (Zhang et al., 2026), Training-Free GRPO (Cai et al., 2025)
Skill storage Bank → single document	SkillRL (Xia et al., 2026), ReasoningBank (Ouyang et al., 2025), SWE-Exp (Chen et al., 2025)	SkillOpt (Yang et al., 2026), ACE (Zhang et al., 2026), Dynamic Cheatsheet (Suzgun et al., 2025)
Generation granularity Batch size 1 → 4	ReasoningBank (Ouyang et al., 2025), ACE (Zhang et al., 2026), Dynamic Cheatsheet (Suzgun et al., 2025)	SkillOpt (Yang et al., 2026), Trace2Skill (Ni et al., 2026), Training-Free GRPO (Cai et al., 2025)

Table 7: Prior self-evolving methods that adopt our choice or the ablated choice for each component in Figure 3.

C.2 Growth and composition of the skill bank

Figure 5 in the main text illustrates the evolution dynamics for Claude-4.6 on `pygments`. With validation, the bank grows through seven accepted updates, reaches 26 items (12 skills and 14 principles) at step 323, and then remains fixed. Without validation, proposed items accumulate until the bank size reaches 453 items (249 skills and 204 principles) by approximately step 400. Despite this much larger bank, the ungated run falls from its early peak to an 80.0% final resolved rate, whereas the compact validated bank retains 92.3%.

Table 3 in the main text extends this comparison to all main runs. Validation retains 204 skills and 192 principles in total, compared with 2,422 skills and 2,024 principles without validation. The gate therefore produces banks that are 11× more compact rather than allowing extracted items to accumulate unconditionally.

D TIME AND EFFICIENCY ANALYSIS

We report total wall-clock time for each main model–repository run. For the component ablations, whose evolution horizons differ by design, we retain the per-step normalization used in the original runtime logs.

D.1 MAIN EXPERIMENT RUNTIME

Table 8 reports total runtime for the main SWE-smith experiments. Averaged across the twelve model–repository runs, ungated evolution takes 50.6 hours, validated evolution takes 122.1 hours, and SkillOpt takes 96.1 hours.

		Train+validation wall-clock time (h)				
Model	**Setting**	**`pygments`**	**`sqlfluff`**	**`deepdiff`**	**`pydicom`**	**Model Avg.**
MiniMax-M2.7	SkillOpt	118.5	134.1	105.7	121.7	120.0
	Evolve w/o val	42.9	46.3	47.2	73.9	52.6
	Evolve w/ val	127.5	107.8	139.0	153.5	131.9
GPT-5.5	SkillOpt	131.9	140.5	90.5	132.6	123.9
	Evolve w/o val	40.1	97.6	55.5	67.3	65.1
	Evolve w/ val	103.9	128.2	101.5	148.8	120.6
Claude-4.6	SkillOpt	51.8	30.5	29.2	66.5	44.5
	Evolve w/o val	32.8	57.6	22.5	23.5	34.1
	Evolve w/ val	126.2	119.3	54.7	154.2	113.6
Repo average	SkillOpt	100.7	101.7	75.1	107.0	96.1
	Evolve w/o val	38.6	67.2	41.7	54.9	50.6
	Evolve w/ val	119.2	118.4	98.4	152.2	122.1

Table 8: Total wall-clock runtime in hours for the main SWE-smith experiments, with held-out test evaluation excluded. The *Model Avg.* column averages repositories within each model, and the *Repo average* block averages models within each repository. MiniMax-M2.7 is served on local GPUs, so its times also reflect local inference throughput.

D.2 ABLATION RUNTIME

Table 9 reports runtime per evolution step for the GPT-5.5 `pydicom` ablations. The variants differ substantially in their work per step: evolving on all tasks is the fastest at 1.14 hours per step, while the sentence-format variant is the slowest among the reported component ablations at 2.41 hours per step.

E CASE STUDY OF USEFUL GENERATED SKILLS

We analyze the final bank from the Claude-4.6 evolution run on `pygments`. Each learned skill or principle is evaluated individually on the same 100-task holdout with three rollouts per task. The reported gain is the change in resolved rate from injecting that item alone relative to the no-memory baseline. "Steps" is the number of adaptation tasks processed when the item entered the bank; equal values indicate items created by the same accepted update.

ID	Component	Configuration	Hours / step
Ref.	Full pipeline	Mixed-only, validate 50	2.11
A1	Skill format	Sentence	2.41
A2	Principle memory	No principles	2.20
A3	Storage	Single document	2.08
A4	Selection	Inject all	1.83
A5	Extraction batch	Batch size 4	1.58
A6	Training stream	All tasks	1.14

Table 9: Wall-clock runtime per evolution step for the GPT-5.5 `pydicom` component ablations. The reference configuration is shaded. Only per-step time is reported; total runtime and the number of evolved steps are omitted.

Table 10: Single-item utility for all items in the final Claude-4.6 `pygments` bank, listed in learning order. Gain is measured relative to the no-memory baseline on the 100-task holdout.

Steps	Item	Type	Category	Gain
1	`scope_reported_vs_scope_tested`	Principle	Repair strategy	+5.33
1	`diff_from_baseline_before_fixing`	Principle	Repair strategy	+9.00
1	`audit_full_modified_file_for_multiple_bugs`	Skill	Repair strategy	+2.00
2	`partial_fix_can_fail_tests_that_full_fix_passes`	Principle	Repair strategy	+5.00
2	`co_introduced_bugs_cluster_in_same_commit`	Principle	Repair strategy	+7.00
2	`verify_fix_completeness_with_full_test_suite`	Skill	Repair strategy	+2.33
2	`git_show_commit_enumerate_all_changes`	Skill	Repair strategy	+8.00
214	`operator_substitution_changes_return_semantics`	Principle	Code semantics	+2.00
214	`boolean_inversion_has_global_side_effects`	Principle	Code semantics	+1.67
214	`test_analyse_text_with_non_matching_input`	Skill	Code semantics	+1.33
217	`deferred_import_requires_empty_initialization`	Principle	Code semantics	+2.00
217	`identical_patch_different_test_outcome_signals_environment_issue`	Principle	Environment	+1.33
217	`audit_set_operations_for_semantic_correctness`	Skill	Code semantics	+1.33
217	`check_kwargs_forwarding_in_super_init`	Skill	Code semantics	+0.33
241	`inverted_condition_implies_inverted_action`	Principle	Code semantics	+1.33
241	`failed_attempt_same_patch_signals_test_environment`	Principle	Environment	+1.00
241	`install_test_dependencies_before_running_tests`	Skill	Environment	+1.67
241	`fix_return_value_alongside_condition_fix`	Skill	Code semantics	+1.33
262	`identical_patch_nondeterministic_outcome_implies_external_factor`	Principle	Environment	+1.67
262	`revert_to_known_good_as_fix_strategy`	Principle	Repair strategy	+9.33
262	`enumerate_all_swapped_values_in_buggy_commit`	Skill	Repair strategy	+10.33

Continued on next page

Table 10 continued

Steps	Item	Type	Category	Gain
262	verify_patch_against_known_good_commit	Skill	Repair strategy	+5.33
323	variable_use_before_assignment_signals_reordered_code	Principle	Code semantics	+3.33
323	none_check_polarity_determines_fallback_path	Principle	Code semantics	+0.33
323	audit_yield_tuple_index_group_consistency	Skill	Code semantics	+2.00
323	trace_all_four_bug_classes_in_single_commit	Skill	Repair strategy	+7.00

The learned skills and principles capture knowledge at different levels of abstraction. Skills generally encode specific, executable repair procedures. For example, audit_yield_tuple_index_group_consistency records a Pygments-specific invariant: a lexer callback must take the token position and value from the same regex capture group. The skill enumerate_all_swapped_values_in_buggy_commit captures a broader workflow for multi-edit regressions: inspect the complete commit, classify every mutation, construct a checklist, and restore every incorrect change.

Principles instead abstract a general diagnostic lesson from a particular failure. variable_use_before_assignment_signals_reordered_code recommends restoring the intended data-flow order rather than masking the symptom with an arbitrary default value. Together, these examples show that evolution acquires both repository-specific operational knowledge and transferable coding principles. We reproduce two detailed skills and one general principle below using the text from the final bank.

audit_yield_tuple_index_group_consistency.md

audit_yield_tuple_index_group_consistency

Scope: Any lexer callback method that yields `(match.start(N), TokenType, match.group(N))` tuples.

Trigger: A bug report or diff shows that match group indices were swapped between `match.start()` and `match.group()` calls in yield statements.

Procedure:

(1) Locate every `yield` statement in the callback that uses `match.start(N)` and `match.group(M)`.

(2) For each yield, confirm that `N == M`—the position index and the group index must refer to the same capture group.

(3) If they differ (e.g., `yield match.start(2), String, match.group(1)`), swap them back so both indices match.

(4) Cross-check against the regex pattern's group numbering to confirm the semantic meaning of each group.

Anti-pattern:

```
# Wrong:  position from group 2, value from group 1
yield match.start(2), String, match.group(1)
yield match.start(1), String, match.group(2)

# Correct:  position and value from the same group
yield match.start(1), String, match.group(1)
yield match.start(2), String, match.group(2)
```

Code Semantics · Skill · evolved steps 323 · gain $+2.00$ *pp.*

enumerate_all_swapped_values_in_buggy_commit.md

enumerate_all_swapped_values_in_buggy_commit

Scope: Functions where a bug-introducing commit swapped or inverted one or more values (booleans, string literals, constants, token types).

Trigger: You find one swapped value (e.g., `'` vs. `"`) in a function and are about to fix only that one.

Procedure:

(1) Run `git show <bad-commit> -- <file>` and collect every − (removed) and + (added) line in the affected function.

(2) For each changed line, categorize the change: swap, inversion, constant change, reorder, deletion.

(3) List all changes as a checklist before writing any fix.

(4) Fix each item on the checklist, not just the first one you noticed.

(5) After fixing, re-run `git show <bad-commit>` and confirm every − line is restored and every + line is removed.

Anti-pattern:

```
# Bug commit changed:
# char = r'"'if double else r"'"(correct)
# to:
# char = r"'"if double else r'"'(wrong)
# AND also changed quantifier '{3,}' to '{2,}'
# AND also changed state_name prefix 't' to 'd'
# Fixing only the char swap leaves the other bugs intact.
```

Repair Strategy · Skill · evolved steps 262 · gain +10.33 pp.

variable_use_before_assignment_signals_reordered_code.md

Use before assignment can indicate reordered code

Statement: When a variable is used before it is assigned in a code block, suspect that lines were reordered or moved during a buggy edit rather than that the variable is simply missing.

When to apply:

- You encounter a `NameError` or a use-before-assignment pattern in code that previously worked.
- A bug-introducing commit touched the ordering of statements in a function.
- The variable is assigned later in the same function, suggesting it was moved rather than deleted.

Better pattern: Trace the full sequence of statements in the function and check whether any assignment was displaced below a use site. Reordering the statements back to their logical order (assign before use) is often the complete fix.

Anti-pattern: Assuming the variable needs to be initialized at the top of the function with a default value, when the real fix is restoring the original statement order.

Code Semantics · Principle · evolved steps 323 · gain +3.33 pp.